The opinion in support of the decision being entered today was *not* written for publication and is *not* binding precedent of the Board

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE BOARD OF PATENT APPEALS
AND INTERFERENCES

Ex parte JAMES R. WASON

Appeal 2007-1186
Application 09/616,809[1]
Technology Center 2100

Decided: June 18, 2007

Before ALLEN R. MACDONALD, JAY P. LUCAS, and
SCOTT R. BOALICK, *Administrative Patent Judges.*

BOALICK, *Administrative Patent Judge.*

DECISION ON APPEAL

Appellant appeals under 35 U.S.C. § 134 from a final rejection of claims 1-3 and 5-17. We have jurisdiction under 35 U.S.C. § 6(b).

We affirm.

---

[1] Application filed July 14, 2000. The real party in interest is International Business Machines Corporation.

STATEMENT OF THE CASE

Appellant's invention relates to a method and system for processing text files used to communicate between computer applications or between a computer application and an end user. (Specification 1:26-30.) In the words of the Appellant:

> [S]pecifically, this invention solves the first problem, describing the text format, by using fragments of text (templates) as overlays for parsing incoming files, or as prototypes to generate segments of output files. There are several important advantages to this approach. The templates are isolated from all other application logic, so it is easy to see why a particular template is producing a resulting text file. The template is a literal image of the text fragment it processes, so it is possible to create the templates from samples of the text file. If the format changes, it is easy to change the corresponding template. For incoming files, there is an important advantage to being able to use a template as a mask to parse the text file. This replaces hard to decipher application logic typically used for parsing.
>
> The second problem of mapping data from the text file to the application is solved by specialized macro classes. These come in two flavors (one for input and one for output). The input macro read in a segment of the text file and use it to initiate application update processing. The output macros derive data from the application and format it into the text file (special format classes are used to describe how to transform the output into the proper shape). This invention provides a basic set of macros, and the facilities to add more as needed.
>
> The solution to the third problem, flow of control, is solved by the interaction of templates and macros. A macro is embedded as a special keyword within a template. When the template reaches that point, it calls the macro. The macro in turn is passed another template name as part of its invocation; as a part of its processing it can invoke that template (which in turn calls

> other macros, etc.). This nested aggregation of templates and macros allows a processing structure to be built up that mirrors the inherit [sic] structure of the text file. Since the behavior of the macro depends both on its internal logic and the template it is passed to invoke, it is possible to reuse the same macro to do different things by passing it a different template. The net effect is that the bulk of the logic needed to describe flow of control is included in the template structure. The template/macro combination is the ultimate expression of the ideal of letting the target file structure determine the structure of the application needed to process it.

(Specification 5:25 to 7:7.)

Claim 1 is exemplary:

1. A method of processing a text file in a computer application, comprising the steps:

forming a plurality of templates having literal fragments of the text file;

providing a macro class to map data from the text file to the computer application;

embedding in one of the templates a pointer to the macro class; and

using said one of the templates as an overlay to parse the text file into segments having data, or as a prototype to generate a segment of an output file;

said using step including the steps of:

i) reaching said pointer in said one of the templates,
ii) when said pointer is reached, using said pointer to invoke said macro class and using said macro class to map data

from one of the segments of the text file to the computer application; and

iii) said macro class then invoking another one of the templates to further process the text file.

The prior art relied upon by the Examiner in rejecting the claims on appeal is:

Andrews	US 6,317,871 B1	Nov. 13, 2001 (filed Jul. 17, 1998)

Claims 1-3 and 5-17 stand rejected under 35 U.S.C. § 102(e) as being anticipated by Andrews.

Rather than repeat the arguments of Appellant or the Examiner, we make reference to the Briefs and the Answer for their respective details. Only those arguments actually made by Appellant have been considered in this decision. Arguments which Appellant could have made but chose not to make in the Briefs have not been considered and are deemed to be waived. *See* 37 C.F.R. § 41.37(c)(1)(vii) (2004).[2]

[2] Except as will be noted in this opinion, Appellant has not presented any substantive arguments directed separately to the patentability of the dependent claims or related claims in each group. In the absence of a separate argument with respect to those claims, they stand or fall with the representative independent claim. *See In re Young,* 927 F.2d 588, 590, 18 USPQ2d 1089, 1091 (Fed. Cir. 1991). *See also* 37 C.F.R. § 41.37(c)(1)(vii).

ISSUE

The issue is whether Appellant has shown that the Examiner erred in rejecting the claims under 35 U.S.C. § 102(e). The issue turns on the interpretation of claim 1 and whether Andrews describes or suggests each and every limitation of the claims.

FINDINGS OF FACT

1. Andrews describes a system for accurately translating computer program code from one high-level computer language to another. (Abstract.) Andrews teaches that "[t]he present invention is embodied in a software system for translating source code to target code where the target code is a different computer language from the source code. The software system is referred to as the 'Rosetta Translator.'" (Col. 5, ll. 44-48.)

2. Andrews teaches that accurate translation of computer languages into a target language is made extremely difficult because of "text preprocessor mechanisms," such as macros, that "cause changes to the text of a program so that the ultimate program is not discernable from a simple 'face value' parsing of the text of the program." (Col. 1, ll. 41-47.)

3. Andrews teaches that macros are translated after they are expanded. (Col. 6, ll. 43-44.) "When a text preprocessor recognizes a reference to a macro, for example 'mac(5),' it substitutes text from the body of the macro at that point." (Col. 4, ll. 32-34.)

4. The system of Andrews "identify[ies] the file structure of a computer program, preserves the file structure of the code after translating source code from one high-level computer language to another, combines pieces of a source file that were generated in different translation sessions, and ensures textual consistency of each piece of generated code in the resultant code files." (Col. 3, ll. 29-35.) This translation "strategy preserves text preprocessor mechanisms such as macros." (Col. 3, ll. 44-45.)

5. One embodiment of Andrews teaches "a method for parsing a source program for translation into a target program of a different language than the source program." (Col. 2, ll. 42-44.) "The method includes the steps of parsing the program to determine whether a line includes a macro definition; creating a first list having a partition associated with a determined macro definition; and using the partition to generate the target program." (Col. 2, ll. 44-49.)

6. Andrews "extracts source language partition templates from a source language code file and assembles target language partition templates into a target language code file." (Col. 7, ll. 11-14.) Partition templates are bounded by, among other things, the "beginning and end

of macro bodies in a macro definition" and the "beginning and end of a macro actual parameter." (Col. 7, ll. 23-24.) Instances of these partition templates are used to populate a fragment tree. (Col. 8, ll. 12-13.)

7. Figure 4 shows an example of partitions in a source language code file. (Col. 8, ll. 8-9.) The source code language file of Figure 4 includes the macro "stuff()."

8. Figures 5 and 6 illustrate the fragment tree that the translator of Andrews builds after scanning the source language code file of Figure 4. (Col. 8, ll. 14-16.) In particular, Figures 5 and 6 show four partitions of the source language code file of Figure 4 in the fragment tree, namely:

   (1) INT glob; DEFINE stuff(a) =

   (2) #; DEFINE junk =

   (3) #; PROC fred; BEGIN

   (4) INT i := 2; i := stuff(i) ; END;

   The contents of the macro body for the macro "stuff()" (i.e., a+13) reside in a macro invocation virtual fragment, and its invocation syntax (i.e., stuff ()) resides in the invoking virtual fragment. (Col. 8, ll. 20-23.)

9. Figure 6 shows that, as part of the macro expansion process, the macro invocation fragment invokes another partition template (labeled part'n) with the macro actual parameters in order to expand the

parameter fragments (i + 13). (Figure 6.) In the macro expansion process, the macro stuff() maps the data which is to be expanded from the source language code file to the translator application. (Figure 6.) To further process the source language code file, the partition template for the parameter fragment is invoked. (Figure 6; Col. 8, ll. 14-39.)

10. The translator of Andrews "pieces together textual representations of instances of target language partition templates to form target language output files, fitting a textual representation of a macro body, for example, into its definition." (Col. 8, ll. 33-36.) "It also pieces together partitions translated only in previous runs with partitions whose translations have just been completed." (Col. 8, ll. 36-39.)

11. "When piecing together partition templates to create a target language code file, the source generator [shown in Figure 2 of Andrews] embeds the macro actual parameters in the partition that contains the invocation syntax for the macro." (Col. 7, l. 66 to Col. 8, l. 2.)

12. "Code generated in different translation sessions can be accumulated into one target language file by merging the partition template maps for different translation sessions and merging the corresponding text for each partition." (Col. 9, ll. 23-25.)

PRINCIPLES OF LAW

Our reviewing court states in *In re Zletz*, 893 F.2d 319, 321, 13 USPQ2d 1320, 1322 (Fed. Cir. 1989) that "claims must be interpreted as broadly as their terms reasonably allow." Our reviewing court further states that "the words of a claim 'are generally given their ordinary and customary meaning.'" *Phillips v. AWH Corp.*, 415 F.3d 1303, 1312, 75 USPQ2d 1321, 1326 (Fed. Cir. 2005) (en banc) (internal citations omitted). The "ordinary and customary meaning of a claim term is the meaning that the term would have to a person of ordinary skill in the art in question at the time of the invention, i.e., as of the effective filing date of the patent application." *Id.* at 1313, 75 USPQ2d at 1326.

Anticipation is established when a single prior art reference discloses expressly or under the principles of inherency each and every limitation of the claimed invention. *Atlas Powder Co. v. IRECO Inc.*, 190 F.3d 1342, 1347, 51 USPQ2d 1943, 1946 (Fed. Cir. 1999); *In re Paulsen*, 30 F.3d 1475, 1478-79, 31 USPQ2d 1671, 1673 (Fed. Cir. 1994).

Under the principles of inherency, a reference anticipates if it necessarily includes or functions in accordance with the claimed limitations. *Atlas Powder Co. v. IRECO Inc.*, 190 F.3d at 1347, 51 USPQ2d at 1946. Inherency may be established by extrinsic evidence, but "[s]uch evidence must make clear that the missing descriptive matter is necessarily present in the thing described in the reference, and that it would be so recognized by persons of ordinary skill." *Continental Can Co. v. Monsanto Co.*, 948 F.2d 1264, 1268, 20 USPQ2d 1746, 1749 (Fed. Cir. 1991). Inherency may not be established by probabilities or possibilities, and "[t]he mere fact that a

certain thing may result from a given set of circumstances is not sufficient." *Id.*

ANALYSIS

Appellant contends that the Examiner erred in rejecting claims 1-3 and 5-17 under 35 U.S.C. § 102(e). Reviewing the findings of facts cited above, we find that the Examiner did not err in rejecting claims 1-3 and 5-17 as being anticipated by Andrews.

Appellant asserts that Andrews does not disclose or suggest "using a macro, which was invoked by one template, both (i) to map data from a text file to a computer application, and (ii) itself to invoke another template to further process the text file," as recited independent claims 1, 6, and 10. (Br. 11; see also Reply Br. 2-4.) We do not agree. Instead, when viewed under the broadest reasonable interpretation of the claims, we agree with the Examiner that Andrews teaches these limitations of the independent claims. (Findings of Fact 1-12; Answer 5-7.)

Appellant argues that, in Andrews, "the macro . . . is not used to map text from a source code to a target code. Instead, the Andrews, et al. procedure involves only substituting a symbol for the macro and its definition." (Reply Br. 2.) According to Appellant, "[t]he Andrews, et al. procedure may use 'macro expansion,' but importantly the macro itself is not used to map text from the source code to the target code." (Reply Br. 3.)

The Examiner replies that "the definition of the word macro alone provides a basis that a macro maps segments of data to a computer

application" and that "Andrews discloses the use of macros for mapping text segments from the text file to the translator application." (Answer 6.)

Although we agree with Appellant that Andrews does not teach the use of a macro to map text from a source code to a target code, the claim language is not so narrow. Specifically, the language of claim 1 is broad enough to encompass macro expansion.

Claim 1 recites "using said macro class to map data from one of the segments of the text file to the computer application." The plain meaning of the claim term "use" is "to put or bring into action or service; employ for or apply to a given purpose." Webster's New World Dictionary Third College Edition 1469 (1994).

Andrews employs macros for the purpose of mapping map text from the source language code file to the Rosetta translator computer application. (Findings of Fact 1, 3-12.) Under a reasonable interpretation of claim 1, the recited "using said macro class" limitation reads on macro expansion performed by the translator of Andrews.

Appellant also argues that "[t]he way in which the first template invokes a macro, which then invokes a second template, is not shown in Andrews." (Br. 13.) The Examiner replies that "Andrews clearly discloses the macro invoking a template during the macro expansion process in order to properly substitute the mapped data of the macro body." (Answer 7.) We agree with the Examiner.

In particular, claim 1 recites "said macro class then invoking another one of the templates to further process the text file." Andrews teaches a macro invocation fragment invoking another partition template as part of the macro expansion process to further process the source code language file.

(Findings of Fact 7-9.) Therefore, this limitation of claim 1 also reads on Andrews.

Claims 2-3, 5-13, and 17 were not argued separately and stand or fall together with claim 1.

Appellant separately argues the patentability of claims 14-16. In particular, Appellant argues that Andrews does not disclose or suggest using "any specific procedure, such as the one set forth in Claim 14, for actually invoking that other template." (Br. 14; Reply Br. 5.) Claim 14 recites that the name of another template is passed to the macro, and the macro then uses that name to invoke that other template. (Br. 14; Reply Br. 5.)

We do not agree with the Appellant on this point either. As previously discussed, Andrews teaches a macro invocation fragment invoking another partition template to further process the source language code file. As part of this process, Andrews discloses passing a name for another template when the macro class is invoked. Specifically, Figure 6 shows the name of the partition template (part'n) invoked by the macro invocation fragment. (Finding of Fact 9.)

Even assuming for the sake of argument that Figure 6 does not expressly disclose passing a name for another template, Andrews inherently discloses passing a name for another template when the macro class is invoked. In order for the macro invocation fragment to invoke the partition template, a person of ordinary skill in the art would recognize that the name of the partition template must necessarily be passed to the macro invocation fragment. (See Findings of Fact 7-9.) Therefore, claim 14 reads on Andrews.

Claims 15 and 16 were argued on the same grounds as claim 14 with respect to this issue, and stand or fall together with claim 14.

CONCLUSION OF LAW

Based on the findings of facts and analysis above, we conclude that the Examiner did not err in rejecting claims 1-3 and 5-17 because Andrews discloses each and every limitation of those claims. The rejection of claims 1-3 and 5-17 is affirmed.

DECISION

The rejection of claims 1-3 and 5-17 is affirmed.

No time period for taking any subsequent action in connection with this appeal may be extended under 37 C.F.R. § 1.136(a)(1)(iv).

AFFIRMED

tdl

Richard L Catania Esq
Scully Scott Murphy & Presser
400 Garden City Plaza
Garden City NY 11530